ESD-TR-92-052 MTR-11201

Compression of Digitized Map Images

By

D.A. Southard

March 1992

Prepared for
AFMSS Program Manager
Electronic Systems Division
Air Force Systems Command
United States Air Force
Hanscom Air Force Base, Massachusetts

Project No. 613B
Prepared by
The MITRE Corporation
Bedford, Massachusetts
Contract No. F19628-89-C-0001

ADA250707

ACKNOWLEDGMENTS

This document has been prepared by The MITRE Corporation under Project No. 613B, Contract No. F19628-89-C-0001. The contract is sponsored by the Electronic Systems Division, Air Force Systems Command, United States Air Force, Hanscom Air Force Base, Massachusetts 01731-5000.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

The Air Force Mission Support System (AF MSS) is a system for automated mission planning. AF MSS will be used by flight crews throughout major commands of the Air Force. The display of digitized navigational charts is central to the function of this system.

BACKGROUND

Chart data will be provided by the Defense Mapping Agency (DMA), as their Equal Arc-second Raster Chart (ARC) Digitized Raster Graphic (ADRG) product. A mission planner will need charts covering an operational area of up to 6,000,000 km^2. The operational area comprises over 200 gigabytes (GB) of ADRG data in its raw form.

The AF MSS specification adopts several strategies for storage reduction:

- *Spatial Reduction.* When displayed, ADRG maps exhibit a magnification factor of about 2.5, compared to the original paper product. The AF MSS specification calls for a spatial reduction factor of 2.
- *Color Quantization.* AF MSS will ultimately run on a variety of equipment, from transportable desk-side systems, to portable laptop units. This represents a large range of cost and performance. The low-end systems will only be able to support 8-bit color look-up table frame buffers with limited color palettes.
- *Image Compression.* The AF MSS specifies storage compression using a vector quantization (VQ) algorithm, or an alternative that meets certain requirements. The mission planning subsystem (MPS) will decompress charts for viewing.

Taken together, these techniques give a 48:1 storage reduction ratio. Some implementations of these techniques, however, will severely degrade the quality of the displayed maps. We investigated alternatives at each step, and various combinations of choices for the entire process.

RESULTS

We prototyped several options at each step, and examined various combinations of options. We tested the algorithms on representative chart types.

Spatial Reduction

We tested several methods of filtering before subsampling map images, including no filtering, neighborhood averaging, and box, triangle, quadratic, and cubic separable filters. Without filtering, the images exhibited sampling artifacts, such as aliasing, stippling, and moiré, that impaired the legibility of these charts. We obtained the best results with the cubic filter.

Color Quantization

We tested several methods for color quantization, including bit slicing, uniform lattice quantization, greedy seeding, popularity, median cut, octree quantization, Linde-Buzo-Gray (LBG) clustering, and pairwise-nearest neighbor (PNN) clustering. For creating customized color tables, octree quantization was fastest, but PNN clustering had the best quality. We recommend the PNN algorithm for selecting customized color tables. We found that the *k*-d tree is a good data structure for implementing efficient color table searches. Color matching should be performed in a perceptually uniform color space.

We found that although (6, 6, 6) uniform lattice color quantization is adequate for quantizing maps *before* compression, much better results will be obtained with custom color tables. Furthermore, the (6, 6, 6) quantization seriously degrades the quality of compressed images, making custom color tables a necessity.

VQ Image Compression

We examined two algorithms for constructing VQ codebooks: LBG and PNN. PNN proved much better than LBG for map compression. We tested three forms of codebook structure: classified, *k*-d tree, and full-search. For image compression, classified VQ was an order of magnitude faster than full-search, but quality was less. Full-search quality can be obtained at half the computational cost by using a *k*-d tree search algorithm for compression. To combine color quantization and VQ compression, we found that it is best to design the codebook and compression algorithms first, then to color-quantize the codebook before map decompression.

We also found that the sequence of events originally envisioned is not practical. The original plan called for 4:1 spatial reduction, 3:1 color reduction, and 4:1 image compression, for a total storage reduction ratio of 48:1. We found that the sequence of events should be 4:1 spatial reduction, then 12:1 image compression. The color reduction has no effect on storage reduction, but may be tailored to the display hardware. We also found that we can attain acceptable quality with higher image compression ratios. This new option opens up more possibilities for cost and performance trade-offs in the AF MSS system design.

SECTION 1

INTRODUCTION

The Air Force Mission Support System (AF MSS) is a system for automated mission planning to be used by flight crews throughout major Air Force commands. The display of digitized navigational charts is central to the function of this system. We report on a series of investigations into compression techniques for use in this program.

1.1 BACKGROUND

Map data will be provided by the Defense Mapping Agency (DMA), as their Equal Arc-second Raster Chart (ARC) Digitized Raster Graphic (ADRG) product [1]. The DMA supplies ADRGs on Compact Disc Read-Only Memory (CD-ROM) media.

The charts are sampled at 100 micron intervals (254 samples per inch). Each sample is a 24-bit color triplet, comprising three 8-bit samples for each primary color: red, green, and blue (RGB). A typical chart, such as a Tactical Pilotage Chart (TPC), has a nominal size of 3'×5'. This represents a storage requirement of almost 400 megabytes (MB) per chart. The different charts vary in the area they cover. A mission planner will require several types of charts covering an operational area of up to 6,000,000 km^2, including:

- Topographic Line Map (TLM)
- Air Target Chart (ATC)
- Joint Operations Graphic (JOG)
- Joint Operations Graphic-Air (JOG-A)
- Tactical Pilotage Chart (TPC)
- Operational Navigation Chart (ONC)
- Jet Navigation Chart (JNC)
- Global Navigation and Planning Chart (GNC)

The operational area can overlap several adjacent charts of each type. The required coverage represents over 200 gigabytes (GB) of data in its raw form.

The storage cost for this amount of data is about $5,000 per GB. Performance and cost goals require a significant storage reduction. Since the cost per GB of disk storage is falling in today's market, we might ask whether we can simply expand disk storage in the future as needed. If information requirements are static, a hardware solution may be appropriate. We anticipate, however, that the information available to mission planners will expand at least as fast as disk technology. An additional problem is that disk controllers can support only a limited number of disks, and data busses can support only a limited number of disk controllers. We believe that our current resources should be conserved as much as possible. Cost and performance factors will continue to make data compression necessary.

The AF MSS specification adopts several strategies for storage reduction:

- *Spatial Reduction.* The pixel spacing on a high-resolution (1280×1024 pixel) 19-inch CRT display is approximately 250 microns. When displayed, ADRG maps exhibit a magnification factor of about 2.5, compared to the original paper product. This is more magnification than needed for legibility on the CRT display. The system specification calls for a spatial reduction factor of 2, so that the displayed image will approximate the original chart size. This factor also allows more of the chart to be viewed on the CRT display. This factor results in a 4:1 reduction in storage size.

- *Color Quantization.* AF MSS will ultimately run on a variety of equipment, from transportable desk-side systems, to portable laptop units. This represents a large range of cost and performance. The low-end systems will only be able to support 8-bit color look-up table frame buffers with limited color palettes. Furthermore, full 24-bit RGB color is more than necessary to display these charts. Combining these two considerations, the specification calls for an 8-bit color representation, which also represents a 3:1 storage reduction.

- *Image Compression.* Chart images contain some redundancy, so that image compression techniques can be effective in reducing storage requirements. The AF MSS specifies a 4:1 storage compression ratio for digitized charts. The mission planning subsystem (MPS) will decompress charts for viewing.

The conjunction of these techniques gives a 48:1 storage reduction ratio. In combination, the steps can interact with each other, and severely degrade the quality of the displayed maps. We investigated alternatives at each step, and various combinations of choices for the entire process. We will describe the alternatives that we explored, the results for each case, and our recommendations.

1.2 IMAGE QUALITY JUDGEMENT CRITERIA

Throughout this paper, we will make recommendations based on our judgement of map image quality. We have not performed quantitative measurements of image quality, but we have established criteria to assist in our subjective evaluations.

- *Background stipple and moiré.* Map areas intended as flat color fields in the paper charts should not exhibit stipple and moiré.

- *Legibility of fine print.* Fine print should remain legible. Examples of fine print include contour line elevation values, vertical obstruction data, and the names of roads, rivers, and streams.

- *Differentiation of rivers and contour lines.* Some processing methods may obscure the difference between rivers and topographical contour lines. These lines should remain distinct.

- *Preservation of symbology.* Map symbols, including airports, vertical obstructions, and navigational symbols must be recognizable.

- *Color Fidelity.* The original paper charts can vary considerably in color rendition, even among adjacent charts of the same type. However, the colors on the computer display should reproduce the colors as closely as possible. Colors that are distinct on the paper chart should be distinct on the computer display.

We inspect portions of the image under magnification. We magnify the image using the pixel replication technique. This procedure ensures that we will observe subtle differences in image quality.

The test data comprised four images selected from four types of maps (geographical area): ONC (California), TPC (mid-western United States), JOG-A (Southern California), and TLM (Germany). The images are 512×512 pixels in resolution. Each pixel is a 24-bit RGB color. Although this is a small sample from the large body of digitized charts available, these images are representative of the types of charts used frequently by mission planners.

1.3 COMPUTING ENVIRONMENT

We used the C programming language for all of the algorithms that we tested. We ran the programs on a single processor of a Silicon Graphics IRIS 4D/340VGX workstation. A single processor on this workstation is rated at 18 SPECmarks. This level of performance is similar to computer configurations proposed for AF MSS. We made no special attempt to optimize our code. We used the same level of automatic compiler optimization on all programs. We measured execution times with the UNIX `time` command.

1.4 SCOPE

This work represents MITRE's technical recommendations for chart image compression. It serves to provide examples of acceptable technical approaches for implementation of certain AF MSS system requirements. The recommendations and opinions presented herein are based on our own investigations and tests. Other methods, which we have not had the opportunity to test, may give performance equal to or better than those we describe.

1.5 ORGANIZATION

In section 2 we describe our investigations into spatial reduction, which includes spatial filtering and image subsampling techniques. Section 3 addresses color quantization methods. We present our findings on image compression algorithms in section 4. Section 5 contains a summary of our results and recommendations.

SECTION 2

SPATIAL REDUCTION

Spatial reduction means reducing the spatial resolution of the chart image. After spatial reduction, the image has fewer pixels in both dimensions. We perform this reduction through a combination of *subsampling* and *filtering*.

2.1 SUBSAMPLING

An image with, say 1024×1024 pixels, can be reduced to an image of 512×512 pixels by selecting every other pixel. This procedure is called subsampling. Subsampling by a factor of s results in an s^2:1 storage reduction ratio. We can subsample with either integral or nonintegral factors. With integral factors, we always select from existing pixel locations. Non-integral factors will result in sampling locations that are in between the existing pixel locations. We need an interpolating function to determine the pixel values for nonintegral pixel positions.

The pixel pitch of the display is about 2.5 times the pixel spacing on the digitized charts. We could select a subsampling factor of 2.5 and get a displayed image that was very close in size to the original chart. This nonintegral factor, however, depends on the specific display device. A slight enlargement of the image is beneficial for legibility on a CRT display. For simplicity, AF MSS specifies an integral factor of two. This factor does not require an interpolating function.

2.2 FILTERING

The ADRG images contain two interesting artifacts, due to the interaction of the characteristics of the original paper product with the digitization process. The maps are printed using half-tone techniques, in which arrays of tiny dots of ink reproduce intermediate tones of color. When digitized with an optical scanning process, the half-toned areas appear *stippled*. We can best describe stipple as a salt-and-pepper texture in the background areas. The cartographer originally intended that these areas appear as a constant shade of color.

The second artifact is an interaction of the half-tone printing screens with the implicit array of optical samples, which produces a *moiré* pattern in these areas. Sometimes the moiré pattern can be quite distinct.

By filtering prior to subsampling, we can eliminate stipple and moiré, and increase the legibility of the subsampled image. We tested several types of filters. This section records our findings.

2.2.1 Impulse

The simplest filter is an *impulse* filter, which is a fancy way of saying no filter at all. We simply select every s^{th} pixel in both dimensions. This procedure leads to poor image quality due to *aliasing*. Aliasing is caused by high spatial frequencies in the original image, which are no longer sampled frequently enough for good reproduction in the output image. High spatial frequencies are needed to reproduce sharp lines and edges. Aliasing causes a jagged appearance in the lettering and exacerbates the moiré pattern effect. The smallest type becomes illegible. Half-toned areas, which were intended to represent areas of constant shading, become noticeably stippled and irregular in appearance. We conclude that a low-pass type filter is required as part of the spatial reduction process.

2.2.2 Ideal Filter

We could filter the image by computing the two dimensional discrete Fourier transform (DFT), multiplying the frequency coefficients by a lowpass transfer function, and then performing the inverse DFT to obtain a filtered image [2]. This procedure is computationally unattractive. Since we intend to subsample the image, it is necessary to filter the image only at the new sample points. We do this by convolving the impulse response of our filter with the image at the new sample points. The impulse response of the ideal lowpass filter take the form of a sinc function. This function has an infinite extent. In practice we must use a finite approximation to the shape of the sinc function. (See figure 1a.)

2.2.3 Neighborhood Averaging

A simple form of low-pass filter is a neighborhood average. We assign weights to pixels surrounding the sampling position and use the weighted average of the surrounding pixels as the value of our sample. We find several forms of low-pass neighborhood average filters in Pratt [3]:

$$\frac{1}{5}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{1}$$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2}$$

$$\frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3}$$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{4}$$

2.2.4 Separable Filters

The remaining filters are in the class of separable filters. A separable filter response function in two dimensions is $F(x,y)=f(x)f(y)$, where x and y represent the horizontal and vertical distances from a sample point. Separable filters can be used for both subsampling and interpolating images. Since they are calculated using analytical functions, it is easy to use integral or nonintegral sampling factors. The following filter functions are illustrated in figure 1.

- *Box*

$$f(x) = \begin{cases} 1 & \textit{if } |x| < \frac{1}{2} \\ 0 & \textit{otherwise.} \end{cases} \tag{5}$$

- *Triangle*

$$f(x) = \begin{cases} 1-|x| & \textit{if } |x| < 1 \\ 0 & \textit{otherwise.} \end{cases} \tag{6}$$

- *Quadratic*

$$f(x) = \begin{cases} 1-2|x|^2 & \textit{if } 0 < |x| < \frac{1}{2} \\ 2(|x|-1)^2 & \textit{if } \frac{1}{2} < |x| < 1 \\ 0 & \textit{otherwise.} \end{cases} \tag{7}$$

- *Cubic*

$$f(x)=\frac{1}{6}\begin{cases}(12-9B-6C)|x|^3+(-18+12B+6C)|x|^2+(6-2B) & \text{if } |x|<1\\ (-B-6C)|x|^3+(6B+30C)|x|^2+(-12B-48C)|x|+(8B+24C) & \text{if } 1\le|x|<2\\ 0 & \text{otherwise.}\end{cases} \qquad (8)$$

  B and C are parameters that define the particular type of cubic spline. Mitchell and Netravali [4] recommend the values (B, C)=(1/3, 1/3).

Separable filters are more efficient than neighborhood averaging filters. A separable filter of width p can be computed in $O(p)$ time, whereas the general case $F(x,y)$ will require $O(p^2)$ time. One computes the filtered image in two passes over the image: one pass for the vertical direction and one pass for the horizontal. The computational requirements increase as the filter shape becomes more sophisticated. In practice, with integral resampling factors, the filter coefficients can be precomputed once and used for all new sample points.

2.3 RECOMMENDATIONS FOR SUBSAMPLING

Neighborhood averaging filters remove stipple and moiré, but the subsampled images are still susceptible to aliasing. The differences between these filters, judged by subsampled image quality, are minor. Of this type of filter, (4) produced the best subsampled image, by a small margin.

The cubic filter performed best of all the filters we tested. The quadratic and triangle filters also produced good images. The quadratic and triangle filters eliminated aliasing, stipple, and moiré, but tended to be a little blurry. The box filter produced the sharpest image, but performed less well at reducing aliasing and stipple. We summarize our results with filters in table 1.

Filtering is essential for good spatial reduction. We benchmarked the computation time required to filter and subsample a 1024×1024 test image. The table shows the relative effort for each filter. The cubic filter is more expensive to calculate, but it produces results much better than the others we tried.

IA9412

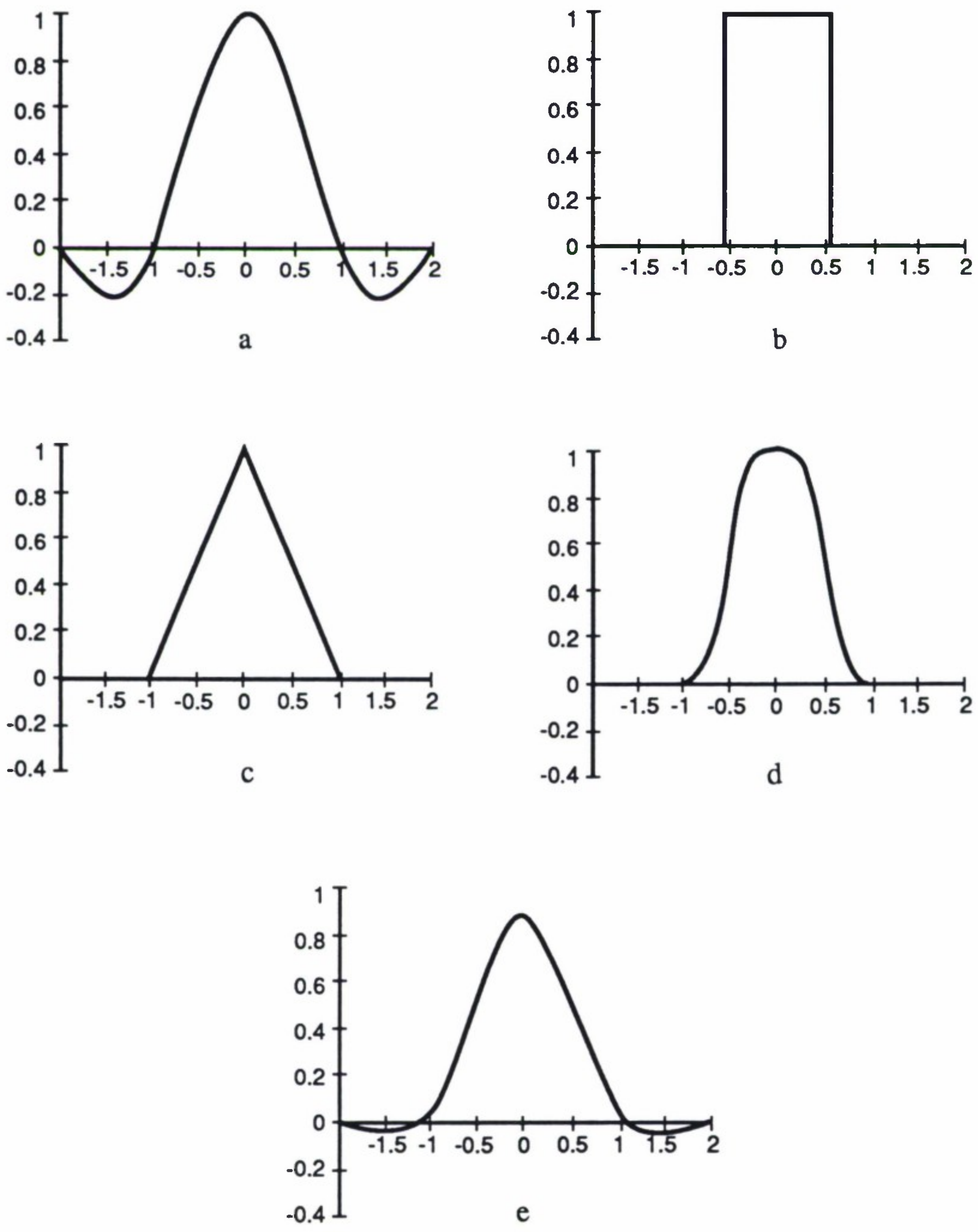


Figure 1. Filter impulse response shapes. (a) Truncated sinc function (b) Box (c) Triangle (d) Quadratic (e) Cubic

Table 1. Comparison of Filtering and Subsampling Methods

Method	Time	Quality
Subsample Only	1.4 sec.	poor
Neighborhood Averaging	8.5	fair
Box	7.9	good
Triangle	10.6	good
Quadratic	10.5	very good
Cubic	16.1	excellent

SECTION 3

COLOR QUANTIZATION

Color quantization means reducing the number of colors used to represent an image. ADRG images are stored as triples of 8 bits each, that is, 24-bit RGB. Combinatorially, this represents about 16.8 million colors. We need only a few hundred colors to display a map image, as we show with the following calculation. Conventional printing uses four colors of ink. That is 16 possible color combinations. If we allow each color 16 levels of shading, for antialiasing lines, letters, *etc.*, we have 256 colors. An 8-bit color lookup table can accommodate this many colors nicely.

3.1 COLOR SPACES

There are many ways to represent color. Due to the physiology of human vision, colors are represented with a three-coordinate tuple. Some coordinates are device-oriented, and some are human-oriented. When we are quantizing 16.8 million colors to 256, or less, the color coordinate system that we use makes a noticeable difference in color fidelity, image contrast, and legibility. Here is a brief review of some prominent color systems:

- *RGB (Red, Green, Blue).* RGB is the most ubiquitous color system in computer graphics. Red, green, and blue are the primary additive colors. RGB coordinates form a color cube, with black at the origin, and white at the opposite diagonal. RGB components correspond directly to the voltages used to drive the electron guns on a CRT display. Our eyes are not equally sensitive to each primary color. It is hard to predict how a change in one component will affect our perception of the color [5].

- *CMY (Cyan, Magenta, Yellow).* Cyan, magenta, and yellow are the subtractive primary colors. CMY is used to describe the relative amounts of pigments used for printing, photography, and color hardcopy. Most commercial color printing uses a four-color process. The fourth color is black (K), so we have CMYK. The interactions of pigments, light, and human vision make it difficult, if not outright impossible, to match colors between a CRT display and a hardcopy, and *vice versa* [5].

- *YIQ (Luminance, In-phase, Quadrature).* YIQ is used in NTSC color television broadcasting. The main consideration in the design of this color system was to maintain compatibility with monochrome TV receivers and to reduce the transmission bandwidth requirements for the chrominance signals, I and Q. Conversion between RGB and YIQ is a linear transformation [5, 6].

- *HSV (Hue, Saturation, Value).* Whereas RGB, CMY, and YIQ are hardware-oriented, HSV is user-oriented. The color components correspond to the intuitive notions of tint (hue: color), shade (saturation: dull to vivid), and tone (value: dark to light). The color space is shaped like a six-sided cone. A cross-section of this space corresponds to the idea of a color wheel [5].

- *HLS (Hue, Lightness, Saturation).* The HLS model is similar to HSV, but forms a double-cone shape [5].

- *CIE XYZ.* The *Commission Internationale de l'Eclairage* (CIE, International Commission on Illumination) established the XYZ tristimulus color space. The primary color components are not real colors, but are mathematical abstractions based on psycho-physical experiments in color vision. CIE XYZ is the international standard for specifying colors precisely [5, 6, 7, 8].

- *CIE L*a*b*.* CIE also established a perceptually uniform color space called L*a*b*, or LAB for short. LAB is a mathematical model that approximates the Munsell color system used by artists and designers. LAB can be determined from the XYZ color coordinates. L* is luminance coordinate, normalized for human perception. a* and b* are the normalized chrominance components [8].

- *CIE L*u*v*.* CIE established another perceptually uniform color space, L*u*v*, or LUV for short. Like LAB, LUV can be determined from the XYZ tristimulus values. There is no solid evidence to suggest which of LAB or LUV is superior to the other [8].

- *HVC (Hue, Value, Chroma).* HVC combines the precision and perceptual uniformity of the LUV standard, with the intuitive nature of HSV and HLS. HVC is based on the CIE LUV color space. It retains the idea of a color wheel, and it uses intuitive variables *value* (dark to light), and *chroma* (dull to vivid) [5, 8, 9].

For color quantization, we want to match 24-bit RGB colors to a smaller set of colors contained in a color lookup table. This involves the idea of measuring color differences.

This is inconvenient to do in cylindrical coordinates systems, so HSV, HLS, and HVC are inappropriate. The measurements should be made in a perceptually uniform color space. Only LAB and LUV qualify. We choose to use LAB based on our own preference. We believe, however, that LUV would be an equally valid choice.

3.2 COLOR QUANTIZATION

We examined several methods of color quantization. Some methods operated in RGB color space, and others in LAB color space. In this section, we describe the techniques that we used, and report our findings.

3.2.1 Universal Color Tables

A universal color table is a subset of color space that can be used to quantize all color images. The advantage of a universal color table is that we need only one color table, no matter how many images we quantize. This is especially important when seaming adjacent charts together, and when displaying several types of charts simultaneously. Some low-end color display systems only have hardware for one 8-bit color lookup table (LUT). High-end workstations generally will support either multiple 8-bit lookup tables, or a larger (*i.e.*, 12-bit) lookup table that can be logically segmented into several smaller tables. A mapping system using a universal color table, then, will apply to a wider cost/performance range of workstation hardware.

3.2.1.1 Bit Slice

The "bit-slice" method operates in RGB color space. We assign some bits to each primary color. We use only the n_c most-significant bits of each color component. We write this as (n_r, n_g, n_b) For a 3:1 reduction, number of bits should sum to eight. Suppose, for example, that we assign three bits to red, three bits to green, and two bits to blue. Then we have a (3, 3, 2) bit slice quantization. To convert a 24-bit color to an 8-bit color table address, we shift each component right $(8\text{-}n_c)$ bits, and concatenate bits to form the address. A color table address comprises three bit-fields, $i_r i_g i_b$, so the color table address $I = (i_r \ll (n_g + n_b)) \mid (i_g \ll n_b) \mid i_b$. (The symbol << stands for a bit shift left; the symbol | stands for a bitwise OR operation.) The corresponding color table entry contains the RGB tuple, $(i_r \ll (8\text{-}n_r), i_g \ll (8\text{-}n_g), i_b \ll (8\text{-}n_b))$.

3.2.1.2 Lattice

The bit slicing method is restricted to 2^{n_c} quantization levels per color component. We can eliminate this restriction by quantizing each axis of the RGB color cube individually. This forms a regular lattice structure of colors. A lattice scheme also can be represented with a three-tuple, (m_r, m_g, m_b). We quantize each component to m_c levels. If $m_r = m_g = m_b = m$, then there will be m neutral colors. If $m_r \neq m_g \neq m_b$, we will have the same tinting problem as the bit slice schemes. The largest uniform lattice with less than 256 colors (8 bits) is (6, 6, 6).

To convert from 24-bit RGB to an 8-bit index to a (6, 6, 6) color table, we first quantize each component separately. We can map the colors using truncation or rounding. For truncation, and assuming integer arithmetic, the quantized component is $i_c = (C(m_c-1))/255$, where C is the component value for R, G, or B. For rounding, the quantized component is $i_c = (C(m_c-1)+127)/255$. For either case, the color table address is $I = i_r+m_r(i_g+m_g i_b)$. The corresponding color table entry contains the RGB tuple $(255i_r/(m_r-1), 255i_g/(m_g-1), 255i_b/(m_b-1))$.

3.2.1.3 Other Universal

The bit-slice and lattice quantization schemes operate in RGB color space. It would be good to have a universal color table that was based on a perceptually uniform color space like LAB. Apparently, this has not been tried before. A drawback is that the color space of each monitor can be different. Some monitors might not be able to reproduce every color in such a universal color table. This idea deserves some future investigation.

3.2.1.4 Results for Universal Color Tables

Bit slicing quantization is easy and efficient to program, but suffers from a defect. There are no 8-bit combinations that will give neutral grey colors. The only neutral colors are black and white. The resulting images tend to be tinted with whichever primaries have the most bits allocated to them. The best scheme was (3, 3, 2), which gives a slight yellow tint to the images. We tested all combinations of 8-bit schemes and confirmed that none was suitable for our mapping application.

AF MSS planned to use the (6, 6, 6) uniform lattice quantization scheme. We tested (6, 6, 6) (216 colors), along with several other nonuniform lattices, including (5, 9, 5) (225 colors), (6, 8, 5) (240 colors), (7, 7, 5) (245 colors), and (6, 7, 6) (252 colors). As we suspected, (6, 6, 6) performed best of this group. Color mapping with truncation darkened the image, and sometimes colors that should be distinct were mapped to the same color table entry. Color mapping with rounding improved this situation considerably. The best technique was to transform the color table and the image into LAB color space, then for each pixel, to search the color table for the entry with the least squared error.

3.2.2 Customized Color Tables

Until now, we have considered only color tables that are universal: they apply equally to all images. These color tables do not provide the best color fidelity possible with an 8-bit color lookup table. As our back-of-the-envelope calculation showed, most maps contain a limited range of colors. A goal of these investigations is to decide whether the AF MSS specification should be amended to allow a customized color table for each chart, or one for each class of charts.

3.2.2.1 Popularity

The popularity algorithm uses a histogram analysis of the image, then selects the M most popular colors. With an 8-bit LUT, for example, we choose $M = 256$. The remaining colors can be mapped to the M popular colors with a least-squared-error approach. Heckbert [10] reports that the success of this algorithm depends on the image in question. Some popular colors might look almost the same. We found this to be true for digitized chart images. It would be better to use one color for all closely related colors, and to reserve space in the color table for colors that look different.

3.2.2.2 Median Cut

The median cut algorithm maps colors so that each color table entry accounts for about the same number of pixels in the image. This algorithm is due to Heckbert [10]. To find the color table, we recursively split the color space into two halves, with equal numbers of colors in each half. Bentley's k-d tree [11] is ideally suited to this algorithm. We then use a nearest neighbor search to quantize image colors. Heckbert reports excellent results with this method. We also found that it works very well for our maps. We used this technique in RGB color space, but the principle would apply equally well, if not better, to a perceptually uniform color space like LAB or LUV.

3.2.2.3 Greedy Seeding

Greedy seeding is a method developed by the author to develop a custom color table. In LAB color space, we sort all the image pixels by luminance (L). We then select a radius *r*, which specifies the maximum perceptual error we are willing to accept. The first pixel in the sorted list becomes the seed for the color table, which initially has only one entry. We examine each pixel, and find the closest match in the color table. If the closest match in the color table is outside the radius *r*, we add this new color to the table. This method gives very good results. It differs from other algorithms in that it limits the maximum perceptual error, instead of the average error. One difficulty, however, is that the number of color table entries varies, depending on the training image and the value of *r*.

3.2.2.4 Octree

Since colors are always three-tuples, we can quantize color space using a three dimensional binary tree, or octree. This method is due to Gervaultz and Purgathofer [12]. The primary advantage of their algorithm is that it is very efficient in both time and memory space requirements. Suppose we want a color lookup table of size *M*. The algorithm works by inserting image pixels into a sparse octree. If we encounter a leaf as we traverse the tree, we average the leaf color with the inserted color. The maximum depth of any leaf in the octree is eight. We insert pixels until the number of leaves exceeds *M*. At this point, we reduce the tree by merging the deepest leaves into their parent node, which becomes a new leaf. A reduction step will decrease the number of leaves up to eight, and add one new leaf in their place, for a net reduction of up to seven. The final color table will have from *M*-6 to *M* colors.

3.2.2.5 LBG Clustering

Selecting a custom color table is similar to a vector quantization (VQ) codebook generation problem. Here our vector size is one pixel, with three color components. We can use the established VQ clustering algorithms to obtain our color table. The Linde-Buzo-Gray (LBG) [13] algorithm has been the standard VQ algorithm for over ten years. This algorithm iteratively improves an initial "guess," until it reaches a local minimum of the distortion measure. The particular result depends on the initial values selected for the guess. We examined three methods for obtaining initial color table values: random seeding, lattice seeding, and splitting. In random seeding, we randomly select *M* colors from the input training image. In lattice seeding, we select *M* values in a uniform lattice quantization. In splitting, we first run the LBG algorithm for a table of length one, then split this table by perturbing each value to create a new entry. We continue clustering, refining, and splitting, doubling the table size each time, until we have the M table entries ($M = 2^i$). We will revisit the LBG algorithm when we discuss VQ image compression.

3.2.2.6 PNN Clustering

The pairwise nearset neighbor (PNN) algorithm is a new algorithm introduced by Equitz [14]. This clustering algorithm uses a k-d tree to partition the training image data. In this case, k=3, for the three color coordinate components. Each node in the k-d tree splits only one coordinate at a time. This splitting is in contrast to the octree algorithm, which splits all three coordinates simultaneously. It also differs in that we fully populate the tree before beginning reduction. Reduction is based on introducing the least squared error. We will examine this algorithm further in the section on VQ compression.

3.2.2.7 The Navy Standard Compressed Aeronautical Chart Database

The Navy uses a custom color table scheme for its digital chart system [15, 16, 17]. This system is installed in AV-8B Harrier and F/A-18 Hornet aircraft. The Navy system uses the Tessellated Spheroid (TS) map projection coordinates. In the TS system, the earth is divided into five zones: north polar, north temperate, equatorial, south temperate, and south polar. The Navy uses six series of charts in each zone. Each zone/series combination has its own color table, for a total of 30 color tables. Thus, each color table is universal to a series of maps within the same zone, but the table for each series/zone combination is determined using a custom color table generation scheme. In practice, some color tables are the same, but 30 distinct sets are possible.

3.2.2.8 Very Small Color Tables

We tried quantizing to a very small set of only 16 colors. This number of colors is available on virtually any personal computer with a color display. None of the algorithms we tested worked well with so few colors. The PNN and median cut algorithms worked best, but below about 64 colors, the map quality really began to suffer. It is surprising, however, that these algorithms worked well with only 64 colors.

We did find that a person can carefully select 16 colors that provide a reasonably good map image. To do this requires knowledge about which colors are significant. Some significant features, such as lakes and streams, and small lettering, make up only a small proportion of the image pixels. Automated algorithms tend to ignore these features in favor of the large areas of color.

3.2.2.9 Results for Customized Color Tables

We tested six algorithms for developing custom color tables: popularity, median cut, greedy seeding, octree, LBG clustering, and PNN clustering. We summarize our results in table 2.

Table 2. Comparison of Results for Custom Color Tables
***N* is the Size of the Image to be Quantized**

Method	Effort for Generation	Quality
Popularity	$O(N \log N)$	poor
Median Cut	$O(N \log N)$	very good
Greedy Seeding	$O(N)$	excellent
LBG Clustering	$O(N)$	poor
Octree	$O(N)$	very good
PNN Clustering	$O(N \log N)$	excellent

Surprisingly, the LBG algorithm performed poorly for color quantization. This result held for any of the three initialization algorithms: random, lattice, or splitting; and for both RGB and LAB color spaces. We believe that this result is because the algorithm does not partition the data. In the refinement step, the LBG algorithm repeatedly takes the centroid (average) color of each cluster. In color perception, it is the relative ratios of the three primary colors that determines our color perception. The centroid of a cluster, while minimizing squared errors, does not preserve the relative ratios of the primaries, so we end with a color that may bear no resemblance to any color in the training image.

The greedy seeding algorithm produces very good results, because it limits the maximum perceptual error, as defined by LAB. To get a predetermined color table length, however, one must adjust the error radius parameter *r* until one obtains the correct number of color table entries.

The octree algorithm gives very good results, even in RGB color space. It is efficient in both time and memory. The desired number of color table entries, however, could be off by as much as six. The resulting set of colors is not optimal. The octree partitioning is arbitrary with respect to the data. One could expect to get better results with an algorithm that partitioned based on the characteristics of the data. The median cut algorithm does just that and gives better results.

The PNN clustering algorithm seems to have the best balance of features. The observed results are excellent. It partitions based on the data characteristics, and it merges based on an optimality criterion. Since the algorithm partitions the data before clustering, the perceived colors remain faithful to the original. Although it is not as fast as the octree algorithm, in return, we get a better result.

3.3 DITHERING

In most applications of image color quantization, we would be well advised to dither the image using the Floyd-Steinberg error diffusion method. Using this method, we calculate the quantization error at each pixel, then distribute an error compensation term among neighboring pixels that have not yet been quantized. As the pixels are quantized, the errors introduced by the quantization are compensated by the surrounding pixels, so that the net error over a region is nearly zero. Dithering trades spatial resolution for color and intensity resolution.

In a general application, dithering with an 8-bit color table can make the quantized image almost indistinguishable from the original at normal viewing distances. But dithering has its price. Dithering adds a high spatial frequency component to an image. Dithering could adversely affect the legibility of fine print and contour lines. It also could interfere with the image compression algorithm. We advise that dithering should not be used for AF MSS. The spatial resolution of a map is more important than the color resolution.

3.4 RESULTS FOR COLOR QUANTIZATION

If we do not consider the effect of color quantization on image compression, we find that (6, 6, 6) lattice quantization is adequate for representing digital maps. As we will soon see, however, the results are different when we take compression into account.

Customized color tables offer better quality map images. One difficulty in the AF MSS arises when adjacent charts, with different custom color tables, must be seamed together. Another is when different chart types must be displayed simultaneously, side-by-side. Some low-end display hardware may not be able to support multiple color tables. The main MSS workstations, however, should support this. If these issues can be addressed, as they were for the Navy system, there is no reason to *dis*allow custom color tables. Furthermore, custom color tables have positive impacts on the image compression algorithms, as we will discuss in the following sections.

The color quantization of map images should be done using a closest match search, performed in a perceptually uniform color space, such as LAB or LUV.

SECTION 4

IMAGE COMPRESSION

In this section we will discuss image compression for the AF MSS.

4.1 VECTOR QUANTIZATION

The term VQ does not uniquely specify an image compression algorithm. There are many forms of VQ [18, 19]. VQ can be combined with other techniques to form hybrid algorithms. Several forms of VQ, along with other compression schemes, have been proposed for compressing digital maps [20, 21]. In the AF MSS, the image must be decompressed very efficiently on a variety of computing platforms. This requirement effectively limits the field to the simple varieties of VQ.

VQ has many advantages in the AF MSS application. For example, for a given codebook, the compression ratio is fixed, so that one can always predict disk usage requirements. Although codebook generation is time-consuming, this need be done only once for a class of images, and then the results can be reused repeatedly. Compared to other image compression techniques, VQ compression is fast. Decompression is even faster, requiring only a series of table look-up operations to regenerate the image.

4.1.1 Clustering Algorithms

In VQ, we divide an image into small units, called *vectors*. The vectors comprise a group of adjacent image pixels. Typically, a vector is a small rectangular grouping of from two to twenty-five pixels. Suppose we have a pre-defined list of vectors, which we call the *codebook*. To compress, we find the entry in the codebook that most closely matches each vector from the input image. We save the index of each entry in the compressed image. To decompress, we simply look-up each index in the codebook and reassemble our image.

We determine a *compression ratio* by dividing the number of bits in each vector by the number of bits required to represent the codebook index. For example, a 2×2 pixel vector requires 4 bytes, or 32 bits; a codebook with 256 entries requires 8 bits for each index. The compression ratio is then 4:1.

The key to VQ is the method by which we obtain a codebook. The codebook determines the quality of the reconstructed image. We find the codebook by a process called *clustering*. In clustering, we analyze one or more *training images* to find a codebook that will minimize a *distortion measure* for the compressed image. The training images represent the class of images to be compressed with the codebook. The distortion measure is usually squared error.

4.1.1.1 Linde-Buzo-Gray Algorithm

The Linde-Buzo-Gray (LBG) [13] algorithm has been the standard VQ algorithm since its introduction in 1980. This algorithm iteratively improves an initial codebook until it reaches a local minimum of the distortion measure. The particular result depends on the initial values selected for the codebook.

There are three basic methods for obtaining initial codebook values: random seeding, lattice seeding, and splitting. In random seeding, we randomly select vectors from the input training image. In lattice seeding, we select values in a uniform lattice quantization of vector space. In splitting, we first run the LBG algorithm for a table of length one, then split this table by perturbing each value to create a new entry. We continue clustering, refining, and splitting, doubling the table size each time, until we have the desired codebook size.

The LBG algorithm is guaranteed to converge after a finite number of iterations, but this number is not predictable *a priori*. The algorithm is optimal in a local sense, but there could be *many* local optima, any of which might be quite different from *the* global optimal codebook.

4.1.1.2 Pairwise Nearest Neighbor Algorithm

The pairwise nearest neighbor (PNN) algorithm was introduced by Equitz [14] in 1989. The fast-PNN variant uses Bentley's *k*-d tree [11] structure for clustering. The idea is to partition a set of training vectors using a *k*-d tree, then at each step to merge the candidate vector pairs that will introduce the least error. Each leaf of the *k*-d tree submits a candidate vector pair. As the clustering proceeds, leaves are merged, and the tree is rebalanced. Eventually, the tree will contain only the desired number of clusters.

4.1.1.3 Results for Clustering Algorithm

Timing results for clustering on 512×512 map images are summarized in table 3. The code vectors are 2×2 color pixels. The PNN algorithm is much faster than LBG. On monochrome images, the compressed image quality is about the same as LBG. For color images, PNN quality is superior to LBG, as we will discuss further in subsequent sections. PNN and LBG are not necessarily exclusive, since a PNN codebook can be used to initialize the LBG algorithm. This practice reduces the number of iterations required for LBG to converge to a local optimum.

Table 3. Comparison of VQ Clustering Algorithms

Chart	LBG	PNN
ONC	4350 sec.	797 sec.
TPC	3101	758
JOG-A	2182	745
TLM	6306	830

4.1.2 Codebook Structure

We can structure the codebook in certain ways to speed up codebook generation and compression.

4.1.2.1 Full Search

In the full-search scheme, we linearly scan the entire codebook for the closest match based on the distortion measure. Each match operation takes $O(M)$ time, where M is the size of the codebook.

4.1.2.2 Classified VQ

In classified VQ, proposed by Ramamurthi and Gersho [22], we divide the codebook into several smaller, specialized codebooks. Each codebook is specialized for a perceptual class of feature: edges, gradients, solids, and mixed. This structure speeds up the clustering by confining searches to a small sub-codebook. With this approach, a matching operation takes $O(M_c)$ time, where M_c is the size of the sub-codebook for class c. It can also improve the compression quality, because human vision is especially sensitive to features like edges, but the squared error distortion measure is not.

For AF MSS, the vectors have four pixels. This length allows a simple classification scheme. We compute the mean value of the pixel brightness for each vector, and compare each pixel to the mean plus a threshold value. The threshold value keeps tiny variations in pixels from interfering with the vector classification. If the pixel is greater than the mean plus threshold, we assign a code of one, otherwise we assign a code of zero. Concatenating the pixel codes as bits gives us a binary number that will take on values of 0 to 15. This exhausts all possible combinations of four pixels.

The only difficulty with classified VQ is in deciding how many codes to assign to each class. We allocated codes based on the percentage of each class within the training image. Although we found this approach to be simple and efficient, it would not be appropriate for a production image compression system. We found that in some classes, a few codes accounted for most of the matched vectors. The classes, to which those codes belonged, tended to be allocated more codes than necessary. It would have been better to allocate more codes to the other classes. Ideally, we want to spread the quantization error evenly over as many codes as possible. We want a code to either match many vectors with little or no error, or a few vectors with moderate error. Ramamurthi and Gersho provide some theoretical guidance, and practical recommendations.

4.1.2.3 Tree Structured

A tree structured codebook is a hierarchical structure [18, 19]. We begin by matching against a small number (say, two or four) of codes. Each code has another set of codes associated with it. Once we find a match, we are confined to select from the children of the selected code. Matches proceed hierarchically until we reach the lowest level of the codebook. A tree structure codebook can speed the matching process by successively narrowing the search as we proceed down the tree structure. The search time is $O(\log M)$. A tree structured codebook arises naturally when we use the LBG splitting technique to develop the codebook.

4.1.2.4 *K*-d tree

Bentley's *k*-d tree structure can be used to perform a nearest neighbor search in $O(\log M)$ time [23]. This gives us the same quality as a full search, but in much less time.

4.1.2.5 Results for Codebook Structure

A structured codebook has a positive impact on data compression. We tested full search, *k*-d tree, and classified approaches. Table 3 summarizes the performance of each algorithm. These times are for the compression of 512×512 pixel color map images, using code vectors of 2×2 color pixels. The codebooks each had 256 codes. The classified VQ algorithm had 15 vector classifications. The computing time of the full search algorithm does not vary with map type. With the other algorithms, the time can vary depending on the distribution of codes in an image. The *k*-d tree algorithm was about twice as fast as full search. The *k*-d tree would be even more effective for larger codebook sizes. The classified VQ algorithm was the quickest, but the quality was not as good as full search or *k*-d tree. We believe that the quality of the classified VQ approach could be improved with some additional work on the allocation of codes to the various classification types.

Table 4. VQ Compression Algorithm Performance

Chart	Full Search	*K*-d Tree	Classified
ONC	131.5 sec.	65.8 sec.	10.5 sec.
TPC	131.3	47.7	10.3
JOG-A	131.5	66.1	10.8
TLM	131.6	57.5	10.4

The structure of the codebook does not affect the efficiency of VQ *de*compression. The time to decompress the 512×512 pixel map images was always 1.3 seconds.

The classified codebook structure can be combined with either LBG or PNN clustering algorithms. Then, we would perform a clustering for each sub-codebook. The combination of PNN and classified codebook structure would dramatically speed up both codebook generation and map image compression.

We did not test the tree-structured method. The tree-structured approach makes sense when the splitting technique is used for LBG codebook development. It is most effective with larger codebooks, between 1024-4096 codes.

4.1.3 Color VQ

VQ is basically a monochrome image algorithm. We must make some additional arrangements to extend it to a color algorithm.

4.1.3.1 Greyscale

We might be tempted take the output from a color quantization step, which is now an 8-bit image, and directly apply VQ to it. This approach will not work very well. The color-mapped image pixels now contain indexes into a color table. The clustering algorithms assume that the pixels are continuous quantities, such as luminance. The clustering algorithms find the centroid, or average value, of a cluster of vectors. It makes no sense to cluster indexes. The decompressed image will have radical color shifts; "average" indexes will point to completely unrelated colors in the color lookup table. We can see now that color quantization must either be integrated into the VQ algorithm, or it must occur *after* VQ.

4.1.3.2 Separate Color Components

The standard approach is to separate the 24-bit RGB image into three images: one image for red, one for green, and one for blue, then perform compression separately on each component image [24]. A variation on this is to transform the image into YIQ color space or another color space that de-correlates the luminance and chrominance components. Then we can use a higher compression ratio on the I and Q components, because human vision is more sensitive to luminance than to chrominance.

The standard approach is not appropriate for AF MSS, because on decompression we still have a 24-bit color image. If we have used YIQ and variable component compression, there will be additional work to reassemble the image and to convert back to RGB for display. This would place an additional, unwanted computational load on the MPS. Another complication is that we might need three codebooks, one for each color component. Finally, the pixels of the compressed image would be a three-tuple of codes. This arrangement is more complicated than we want. We want to decompress directly to an 8-bit color lookup table image.

4.1.3.3 Combined Color Components

Another approach is to include the three color components in the vector. Suppose a pixel block has dimensions $w \times h$, where w is the block width and h is the block height. A $w \times h$ pixel block has a vector length of $3wh$. Now we can see that the 2x2:1 compression called for in the AF MSS specification is really a 12:1 compression. On decompression, we will have a 24-bit color image. There will be, however, a maximum of only Mwh unique colors. In essence, the VQ algorithm already performs a color quantization. If $M = 256$, for example, there will be 1024 unique colors in the decompressed image.

4.1.3.4 Color Mapped

It still takes 24-bits to represent our image. We want to have an 8-bit color-mapped image. We can get this by color-quantizing the decompressed image. However, since all the colors in the decompressed image must originate from the VQ codebook, we can quantize the codebook ahead of time. We call this approach color-mapped VQ. This achieves our goal of a VQ algorithm that decompresses directly to an 8-bit color image.

4.1.3.5 Results for Color VQ

On color images, we find that LBG has serious shortcomings. As we stated in the color quantization section, LBG tends not to preserve the relative ratios of the color components, so the colors can come out looking quite different than intended. This results in serious legibility problems for maps with fine print. This applies especially to ONC and JOG-A series charts. The PNN algorithm, because it partitions the data first, does not suffer from this problem.

We tested several methods for integrating color quantization with VQ. We must always quantize at least once after VQ, since clustering will modify the selection of colors in the codebook. We also can quantize the colors before clustering and during clustering. If we quantize before clustering, the VQ algorithm will design a codebook for the quantized image. If we quantize during clustering, we coerce intermediate codebook results to our color table. This decreases the number of iterations required by the LBG clustering algorithm. None of these options, however, has any beneficial effect on image compression. It is best to design the VQ codebook on the original training image, then perform color quantization afterward.

A difficulty with color mapped VQ is that many VQ codes may be only slightly different. Color quantization using the (6, 6, 6) lattice quantization can map many code vectors to the same set of values. This results in duplicate entries in the codebook. Duplicate entries are useless codes. They reduce the fidelity of VQ compression. An exception to this observation is in the classified VQ or tree-structured approaches, where it is acceptable to have duplicate codes in different classes or subtrees, but not in the same class or subtree.

The useless code problem can be virtually eliminated by using a custom color table. A custom color table, which is itself a product of clustering, will match the codebook colors very closely.

4.3 RESULTS FOR VQ COMPRESSION

This section combines all our observations for image compression. Figure 2 illustrates the results we describe.

4.3.1 Clustering Algorithm

Although the LBG clustering algorithm works well on monochrome or separated color component images, we find that it has serious shortcomings when applied to combined component color images. Combined component VQ, however, is essential for efficient decompression. We advise the use of the fast-PNN algorithm for constructing VQ codebooks.

4.3.2 Codebook Structure

The *k*-d tree searching mechanism proved to be an effective way to speed up image compression, and maintain quality. Matching should be performed in a perceptually uniform color space, such as LAB or LUV.

4.3.3 Color VQ

We found that color quantization must take place *after* VQ. This is implemented as a 12:1 combined color component VQ compression, followed by color quantization of the codebook. Color matching should be performed in a perceptually uniform color space, such as LAB or LUV. Uniform lattice (6, 6, 6) color quantization degrades the quality of the decompressed image. Custom color tables, which also can be constructed with the PNN algorithm, are much better at retaining image quality.

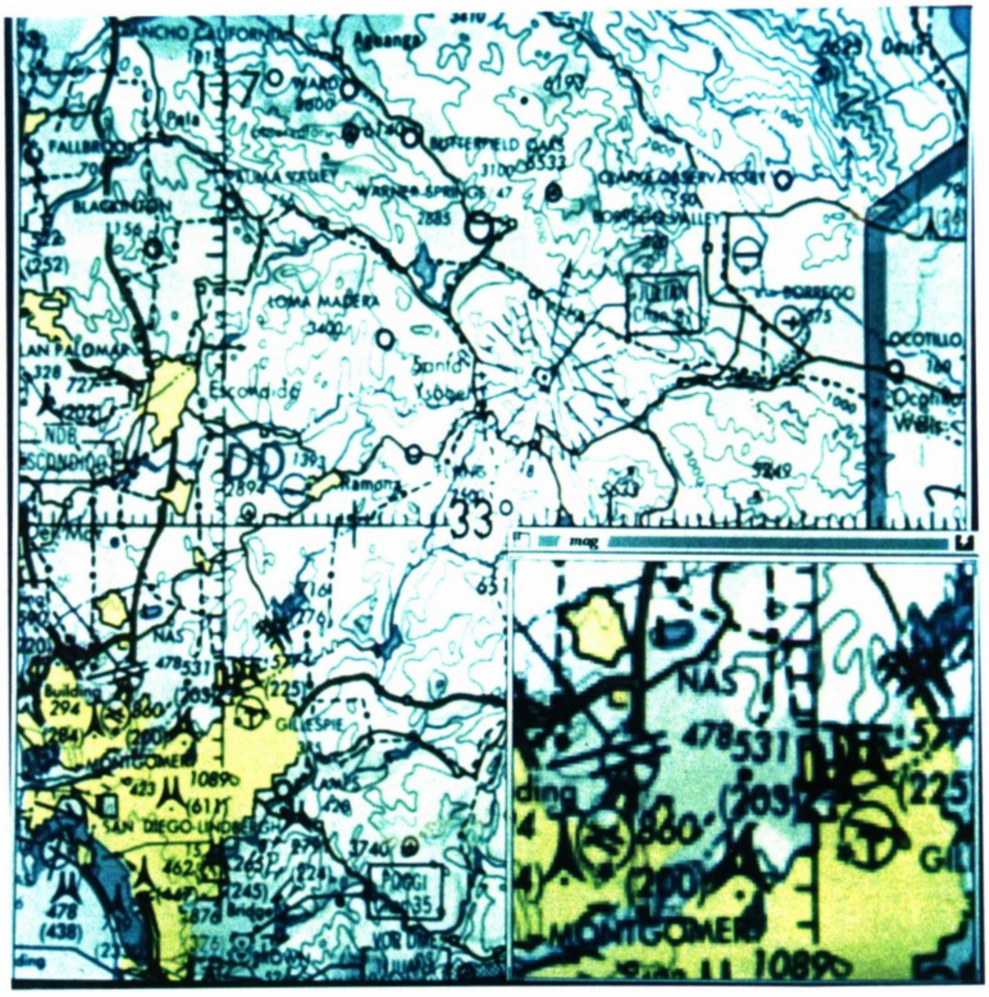


Figure 2. VQ Compression of Digitized Chart Images
(a) Original ONC (Spatially Reduced)

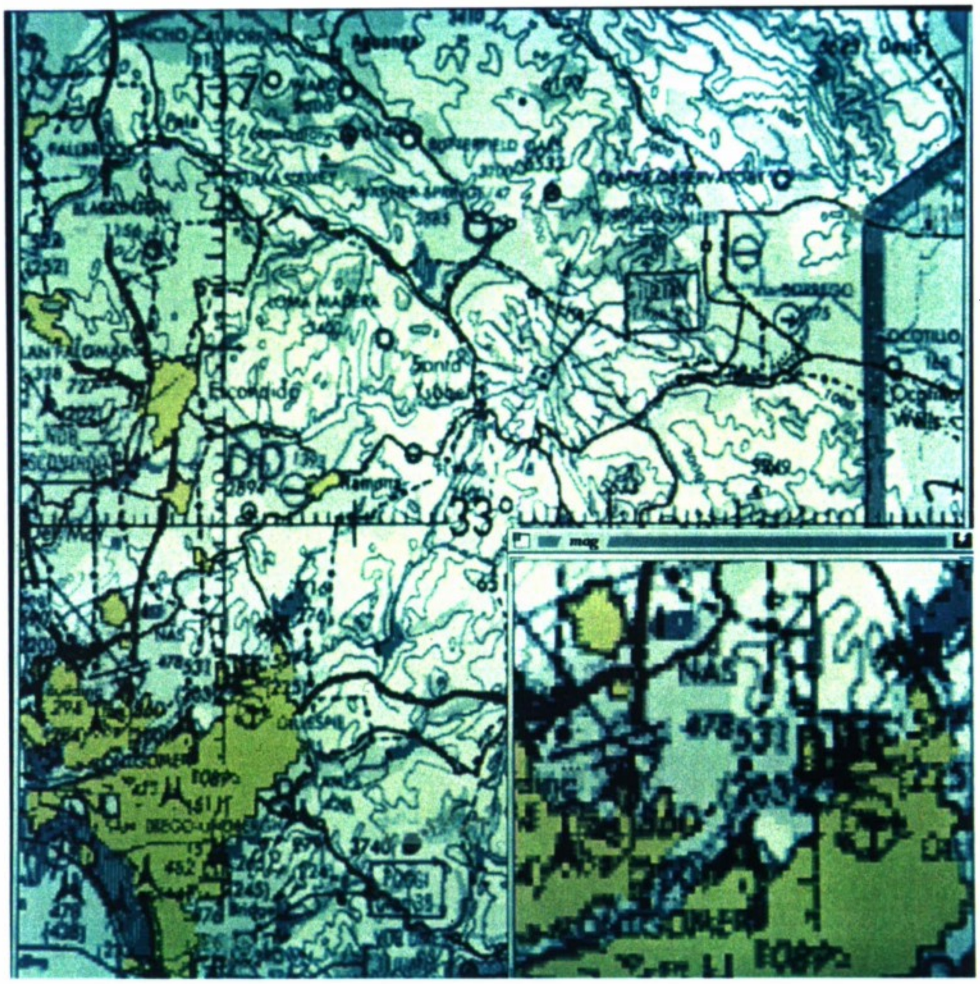


Figure 2 (continued)
(b) 12:1 Compression with LBG Codebook and (6, 6, 6) Color Table

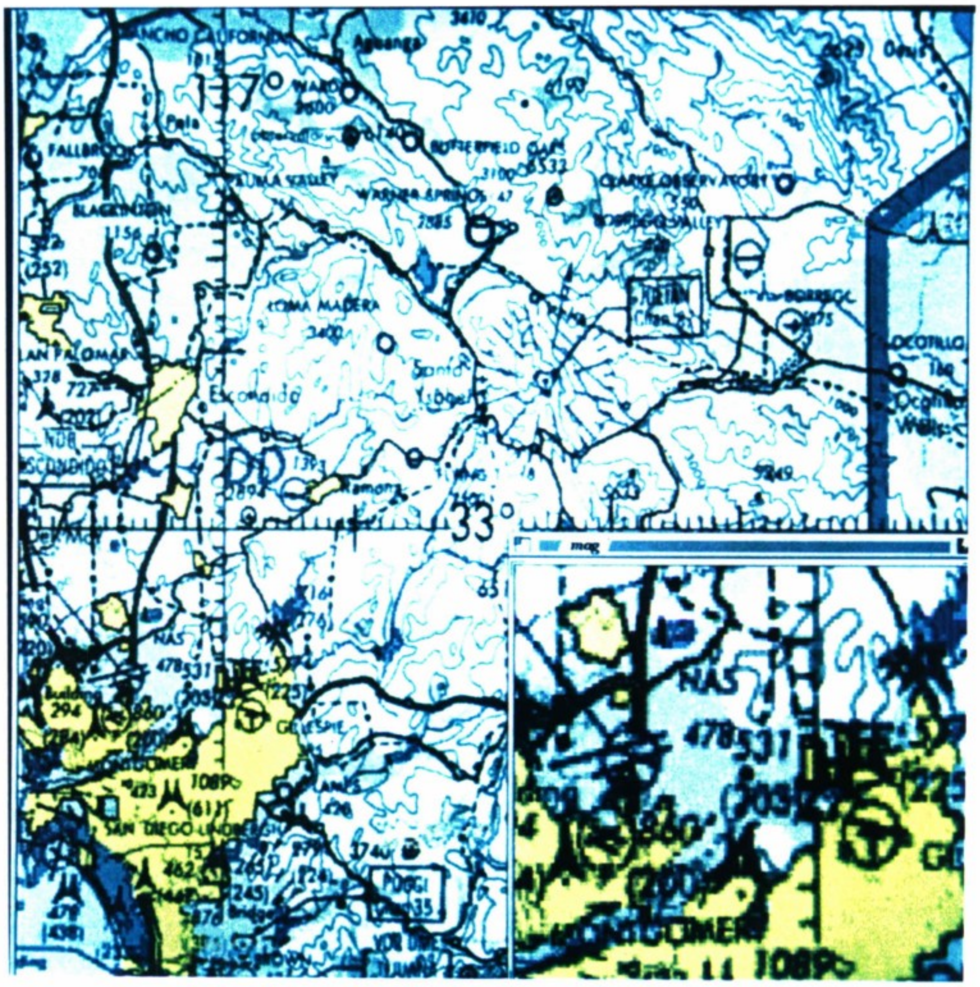


Figure 2 (continued)
(c) 12:1 Compression with PNN Codebook and Custom Color Table

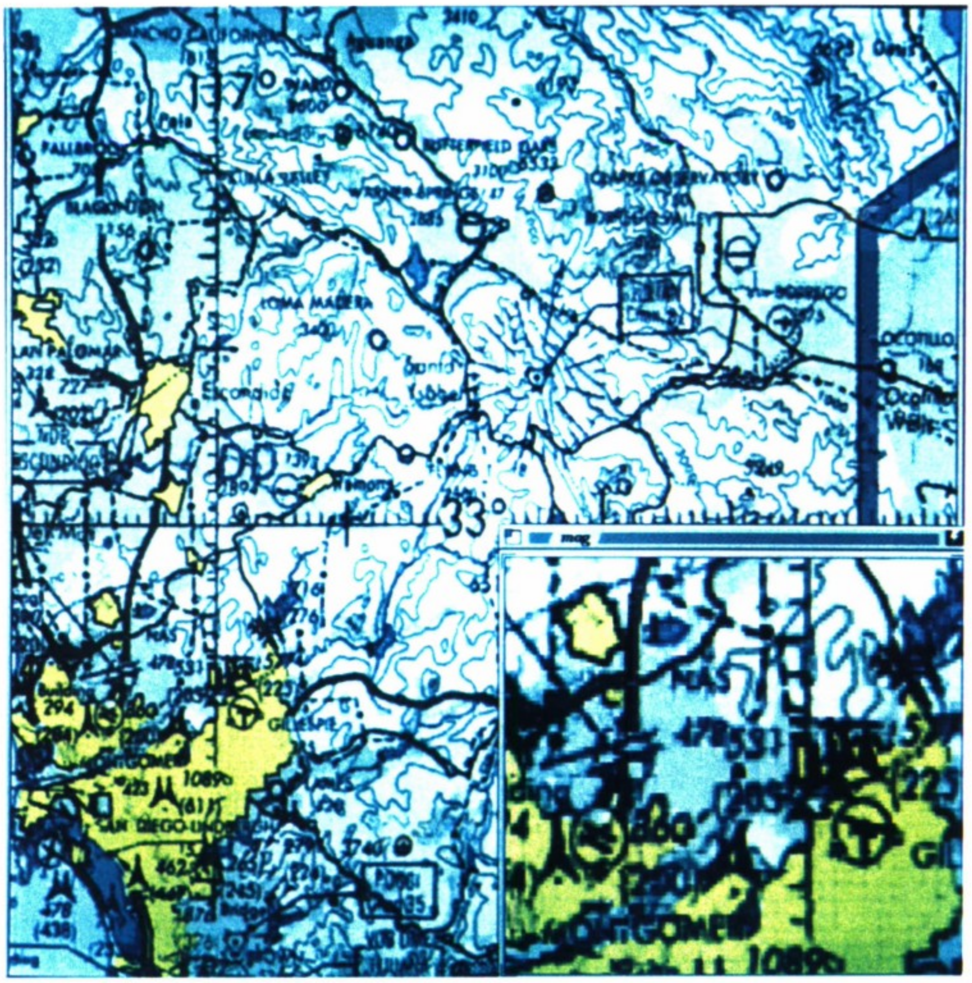


Figure 2 (concluded)
(d) 32:1 Compression with PNN Codebook and Custom Color Table

4.3.4 Alternate Compression Ratios

We experimented with parameters for VQ compression. Image quality is usually in inverse proportion to compression ratio. We found, however, that higher compression ratios often worked as well as the 12:1 needed for AF MSS. (See figure 2.) For example, a 4×4 combined component vector (384 bits) with a codebook size of 4096 codes (12 bits) has a compression ratio of 32:1. This 32:1 compression looked as good as the 12:1 compressed imagery. It appears that the small codebook size, 256 in the 12:1 compression scheme, is too small to take full advantage of VQ compression. A larger vector and codebook allows the clustering algorithm more degrees of freedom for tuning the codebook. There are trade-offs, however. Larger vector sizes and codebooks require more time for compression, and more memory space for decompression. The codebook indexes will no longer fit neatly into 1 byte. In practice, a 12 bit code may have to be stored in 2 bytes (16 bits). This design eliminates computational overhead for bit-packing and unpacking, but decreases the effective storage compression ratio to 24:1. Nevertheless, higher compression is an option for AF MSS.

4.3.5 VQ of Satellite Imagery

The VQ compression techniques used for digitized chart images work well for monochrome satellite images, too. We compressed a SPOT image at two levels of compression: 2×2 vectors with 256 codes for a ratio of 4:1, and 4×4 vectors with 4096 codes for a ratio of 10.67:1. We judged the higher compression ratio to have better quality. The 4:1 compressed image showed more compression artifacts. The short vector size and codebook length simply do not take full advantage of the strengths of VQ. Figure 3 shows VQ applied to satellite imagery.

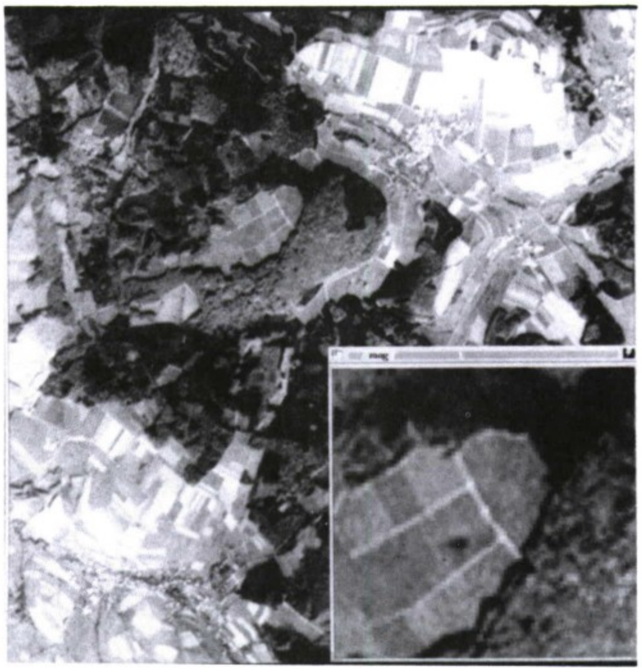
4.4 IMPLICATIONS

A surprising result is that the color quantization has no effect on image compression. The compression is completely determined by the vector size and the codebook length. Consequently, it is possible to color-quantize the same codebook in a different way for each display hardware configuration. For example, on desk-side workstations that support 24-bit RGB frame buffers, color quantization may not be necessary. On a portable or laptop computer, the codebook can be transformed to work with an 8-bit color mapped frame buffer. In this way we can have the best quality supported by the hardware.

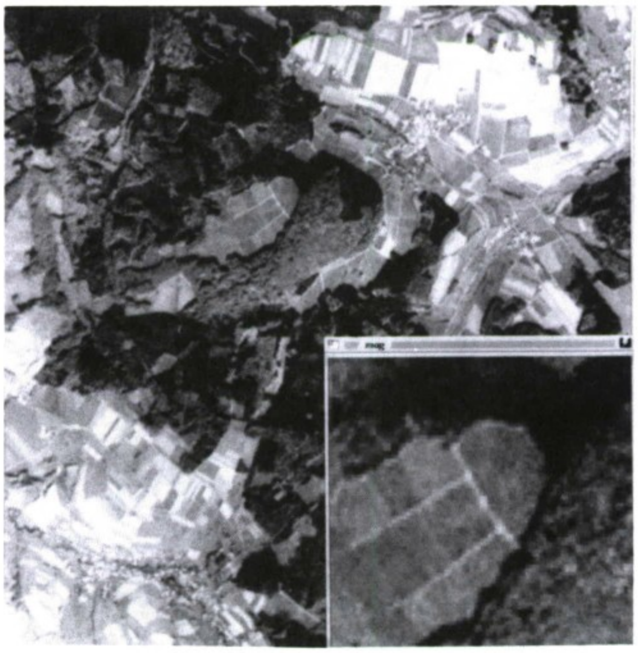
Using a PNN customized color table, we found that we got reasonable map renditions with as few as 60 color table entries. This suggests two possible solutions for the multiple color table problem:

- Use a different custom color table for each chart. If two or more charts must be displayed simultaneously, use a reduced color table for each chart, so that the total number of color table entries remains less than or equal to 256. To provide this kind of flexibility, it would be necessary to store several versions of each color table with each chart. For example, one with 60 entries, one with 120, and one with 240, reserving 16 entries for system use. Only one version of the codebook would be required. It would be stored in full 24-bit RGB, so that the system could select the appropriate color quantization level as needed.

- Use a single custom color table for an entire series of charts. This was the approach adopted in the Navy system, and it is probably the most practical alternative. If adjacent charts are not radically different, 256 colors should be enough for a whole series of charts of the same type, for example, all TPCs in the same ARC zone.

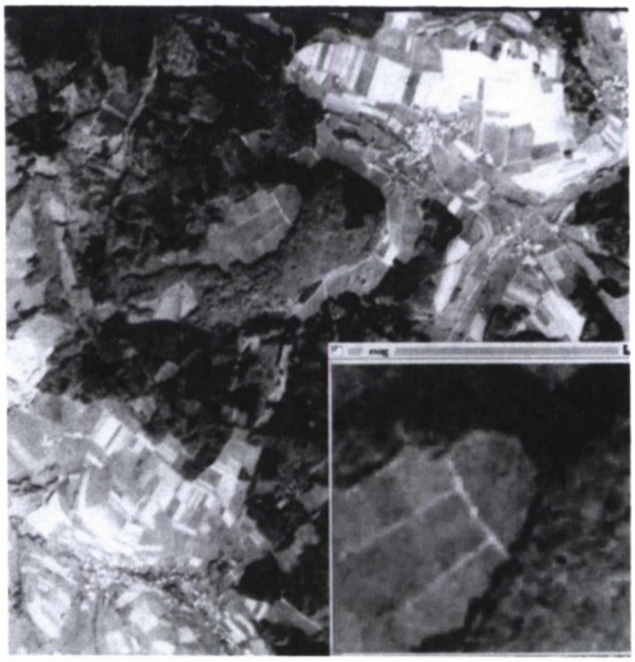


(a) Original



(b) 4:1 Compression (PNN)

Figure 3. VQ Compression of SPOT Satellite Image



(c) 10.67:1 Compression

Figure 3 (concluded)

SECTION 5

CONCLUSION

In this section, we summarize our results and discuss a few topics for further consideration.

5.1 SUMMARY OF FINDINGS

The AF MSS specification calls for three steps to reduce digitized map storage requirements:

- *Spatial reduction*
- *Color quantization*
- *Image compression.*

For spatial reduction, we tested several methods of filtering prior to subsampling map images, including: no filtering, neighborhood averaging, and box, triangle, quadratic, and cubic separable filters. We found that filtering was essential to maintain image quality. We obtained the best results with Mitchell and Netravalli's cubic filter [4].

We tested several methods for color quantization, including bit slicing, uniform lattice quantization, popularity, median cut [10], greedy seeding, octree quantization [12], LBG clustering [13], and PNN clustering [14]. For creating customized color tables, octree quantization was fastest, but PNN clustering had the best color quality. Color matching should be performed in LAB or LUV color space. The *k*-d tree [11, 23] is an excellent data structure for implementing efficient color table searches.

We examined two algorithms for constructing VQ codebooks: LBG and PNN. PNN proved much better than LBG for combined color component map compression. We tested three forms of codebook structure: full-search, *k*-d tree [23], and classified [22]. For image compression, classified VQ was much faster than full-search or *k*-d tree, but quality was less. Full-search quality can be obtained at half the computational cost by using a *k*-d tree search algorithm for compression. To combine color quantization and VQ compression, we found that it is best to design the codebook and compression algorithms for combined color component compression, then color-quantize the codebook once before decompression.

Although (6, 6, 6) uniform lattice color quantization is adequate for quantizing maps *before* compression, it seriously degrades the quality of compressed images, making custom color tables a necessity.

5.2 OTHER TOPICS

In this study, we have largely confined ourselves to looking at options that remained within the general framework of the AF MSS specification. We were primarily concerned with ways to implement these specifications. The AF MSS specification permits other types of compression schemes if they are nonproprietary, standard, and meet certain performance requirements. We briefly discuss some of these possibilities.

- *Discrete Cosine Transform (DCT).* A form of DCT image compression has been standardized in the Joint Photographic Experts Group (JPEG), a working group of the International Standards Organization (ISO). As a standard algorithm, JPEG is very attractive. Integrated circuit manufacturers are building chips sets that execute the JPEG algorithm in specialized hardware. At this time, however, standardized image compression boards are not available for the full range of computing platforms envisioned for AF MSS. Decompression in software with JPEG is more computationally intensive than VQ. The compression ratio for JPEG varies depending on image content. We do not think that JPEG is currently an option for AF MSS.

- *White/Los Alamos Algorithm* [25]. We have recently learned that Dr. James White of Los Alamos National Laboratories has developed a color quantization and clustering algorithm for compression of satellite imagery. This algorithm produces a compressed image that is easily manipulated by personal computers. The algorithm sounds ideal for AF MSS. Apparently, Los Alamos plans to patent this system. As of this writing, detailed information is not available. We will keep abreast of these developments.

- *Lossless Compression* [26, 27]. As an alternative to VQ, some schemes suggest performing the color quantization first, then using a *lossless* image compression technique, such as run-length coding or quadtree coding. The color quantization effectively increases the length of runs, and of homogeneous regions, making the quantized images ideal candidates for these forms of compression. We found that with run-length coding, compression ratios were variable, and they were substantially less than VQ.

LIST OF REFERENCES

1. Defense Mapping Agency, April 1989, *Product Specifications for ARC Digitized Raster Graphics (ADRG)*, DMA Aerospace Center, St. Louis, MO.

2. Gonzalez, R. C. and P. Wintz, 1987, *Digital Image Processing*, 2nd ed., Reading, MA: Addison-Wesley, pp. 163-173.

3. Pratt, W. K., 1978, *Digital Image Processing*, New York: John Wiley & Sons, pp. 319-321.

4. Mitchell, D. P. and A. N. Netravali, August 1988, "Reconstruction Filters for Computer Graphics," *Computer Graphics (Proc. SIGGRAPH)*, Vol. 22, No. 4, pp. 221-228.

5. Foley, J. D., A. van Dam, S. K. Feiner, and J. F. Hughes, 1990, *Computer Graphics: Principles and Practice*, Reading, MA: Addison-Wesley, pp. 547-600.

6. Hall, R., 1989, *Illumination and Color in Computer Generated Imagery*, New York: Springer-Verlag, pp. 221-240.

7. Meyer, G. W., 1986, "Tutorial on Color Science," *The Visual Computer*, Vol. 2, pp. 278-290.

8. Wyszecki, G. and W. S. Stiles, 1982, *Color Science: Concepts and Methods, Quantitative Data and Formulae*, 2nd ed., New York: John Wiley & Sons, pp. 164-169.

9. Murch, G. M. and J. M. Taylor, July 1988, "Sensible Color," *Computer Graphics World*, pp. 69-72.

10. Heckbert, P., July 1982, "Color Image Quantization for Frame Buffer Display," *Computer Graphics (Proc. SIGGRAPH)*, Vol. 16, No. 3, pp. 297-307.

11. Bentley, J. L., September 1975, "Multidimensional Binary Search Trees Used for Associative Searching," *Commun. ACM*, Vol. 18, No. 9, pp. 509-517.

12. Gervautz, M. and W. Purgathofer, 1990, "A Simple Method for Color Quantization," In: A.S. Glassner, ed., *Graphics Gems*, San Diego, CA: Academic Press, pp. 287-293.

13. Linde, Y., A. Buzo and R. M. Gray, January 1980, "An Algorithm for Vector Quantizer Design," *IEEE Transactions on Communications*, Vol. COM-28, No. 1, pp. 84-95.

14. Equitz, W. H., October 1989, "A New Vector Quantization Clustering Algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, pp. 1568-1575.

15. Lohrenz, M. C. and J. E. Ryan, March 1990, "The Navy Standard Compressed Aeronautical Chart Database," Interim Report, Naval Oceanographic and Atomospheric Research Laboratory, Code 351, Stennis Space Center, MS.

16. Lohrenz, M. C., P. B. Wischow, H. Rosche III, M. E. Trenchard, and L. M. Riedlinger, March 1990, "The Compressed Aeronautical Chart Database: Support of Naval Aircraft's Digital Moving Map Systems," *IEEE PLANS '90: Position Location and Navigation Symposium*, pp. 67-73.

17. Shaw, K. B., J. E. Ryan, M. C. Lohrenz, M. G. Clawson, L. M. Riedlinger, and J. I. Pollard II, December 1989, "The NORDA MC&G Map Data Formatting Facility: Development of a Digital Map Data Base," NORDA Report 233, DTIC AD-A220 508, Naval Ocean Research and Development Activity, Stennis Space Center, MS.

18. Abut, H. (ed.), *Vector Quantization*, New York: IEEE Press, 1990.

19. Rabbani, M. and P. W. Jones, 1991, *Digital Image Compression Techniques*, Bellingham, WA: SPIE Optical Engineering Press.

20. Barad, H. and A. B. Martinez, December 1989, "Final Report: Digital Map Research," Technical Report 89-22, Signal & Image Processing Lab, Electrical Engineering Department, School of Engineering, Tulane University, New Orleans, LA.

21. PAR Government Systems Corp., July 1990, *Cartographic Data Compression/ Decompression Study*, RADC-TR-90-112, Vol. 1, Rome Air Development Center, Griffiss Air Force Base, NY.

22. Ramamurthi, B. and A. Gersho, November 1986, "Classified Vector Quantization," *IEEE Transactions on Communications*, Vol. COM-34, No. 11, pp. 1105-1115.

23. Friedman, J. H., J. L. Bentley, and R. A. Finkel, September 1977, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Math. Softw.*, Vol. 3, No. 3, pp. 209-226.

24. Murakami, T., K. Asai, and A. Itoh, April 1986, "Vector Quantization of Color Images," *Proc. IEEE Conf. Acoust., Speech, Signal Processing*, Vol. 1, pp. 133-136.

25. Brown, C., 27 May 1991, "Algorithm unlocks Landsat data," *Electronic Engineering Times*, pp. 39 & 43.

26. Potlapalli, H., M. Y. Jaisimha, H. Barad, A. B. Martinez, M. C. Lohrenz, J. Ryan, and J. Pollard, March 1989, "Classification Techniques for Digital Map Compression," *Proc. 21st Southeastern Symposium on System Theory*, Tallahassee, FL, pp. 268-272.

27. Jaisimha, M. Y., H. Potlapalli, H. Barad, A. B. Martinez, M. C. Lohrenz, J. Ryan, and J. Pollard, April 1989, "Data Compression Techniques for Maps," *Proc. IEEE 1989 Southeastcon*, Charleston, SC, pp. 878-883.

GLOSSARY

ADRG	ARC digitized raster graphic
AF MSS	Air Force Mission Support System
ARC	equal arc-second raster chart
ATC	Air Target Chart
C	color component value
CD-ROM	compact disc—read only media
CIE	*Commission Internationale de l'Eclairage* (International Commission on Illumination)
CMY	cyan, magenta, yellow
CMYK	cyan, magenta, yellow, black
CRT	cathode-ray tube
DCT	discrete cosine transform
DFT	discrete Fourier transform
DMA	Defense Mapping Agency
DPS	data preparation subsystem
$f(x)$	separable filter response function
GB	gigabyte (2^{30} bytes)
GNC	global navigation and planning chart
HLS	hue, lightness, saturation
HSV	hue, saturation, value
HVC	hue, value, chroma
i	integer value, or value of a bit field
I	index of a color look-up table entry
ISO	International Standardization Organization
k	number of dimensions of a vector
JNC	jet navigation chart
JOG-A	joint operations graphic—air
JOG-G	joint operations graphics—ground
JPEG	Joint Picture Experts Group

LAB	CIE L*a*b* perceptually uniform color space
LBG	Linde-Buzo-Gray (clustering algorithm)
LUT	lookup-up table
LUV	CIE L*u*v* perceptually uniform color space
m	number of quantization levels
M	number of entries in a color look-up table
MB	megabyte (2^{20} bytes)
MPS	mission planning subsystem
n	number of bits
N	number of pixels in an image
NTSC	National Television Standards Commission
ONC	operational navigation chart
p	width of filter neighborhood
PNN	pairwise nearest neighbor
r	perceptually radius in LAB or LUV color spaces
RGB	red, green, blue
s	subsampling rate
SPEC	System Performance Evaluation Cooperative
SPOT	*Satellite Pour l'Observation de la Terre* (Earth Observation Satellite)
TLM	topographic line map
TPC	tactical pilotage chart
TS	tessellated spheroid
VQ	vector quantization
XYZ	CIE tristimulus color coordinates
YIQ	luminance, in-phase, quadrature (used in NTSC color television broadcasting)